SETTINGSAPP

Code analysis

By: Gateway

2022-03-29

CONTENT

INTRODUCTION

This document contains results of the code analysis of SettingsApp.

CONFIGURATION

- Quality Profiles
  - Names: Sonar way [Java]; Stryker [Kotlin]; Stryker [XML];
  - Files: AX9JLBXVZWz9exVyjg9p.json; AX_WRsJAZoNvhpLrXfw9.json; AX_WbTE_ZoNvhpLrXf1b.json;
- Quality Gate
  - Name: Sonar-StrykerAndroid
  - File: Sonar-StrykerAndroid.xml

SYNTHESIS

ANALYSIS STATUS

Reliability	Security	Security Review	Maintainability
A	A	E	A

QUALITY GATE STATUS

Quality Gate Status	Failed

Metric	Value
Reliability Rating on New Code	OK
Security Rating on New Code	OK
Maintainability Rating on New Code	OK
Duplicated Lines (%) on New Code	ERROR (14.0% is greater than 3%)

METRICS

Coverage	Duplication	Comment density	Median number of lines of code per file	Adherence to coding standard
0.0 %	7.1 %	6.9 %	24.0	99.5 %

TESTS

Total	Success Rate	Skipped	Errors	Failures
0	0 %	0	0	0

DETAILED TECHNICAL DEBT

Reliability	Security	Maintainability	Total
-	-	1d 0h 7min	**1d 0h 7min**

METRICS RANGE

	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	0.0	0.0	0.0	0.0
Max	1049.0	1082.0	14084.0	99.1	0.0	45.2

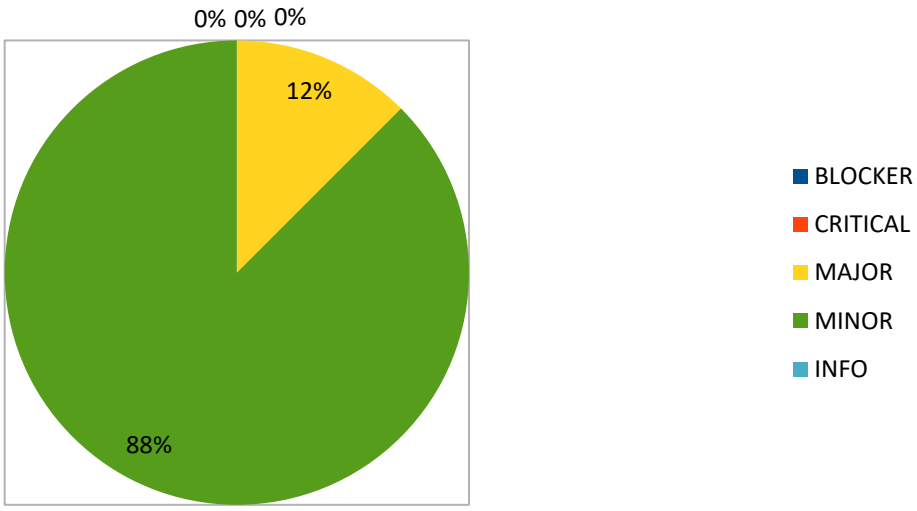
VOLUME

Language	Number
Java	629
Kotlin	9139
XML	4316
Total	14084

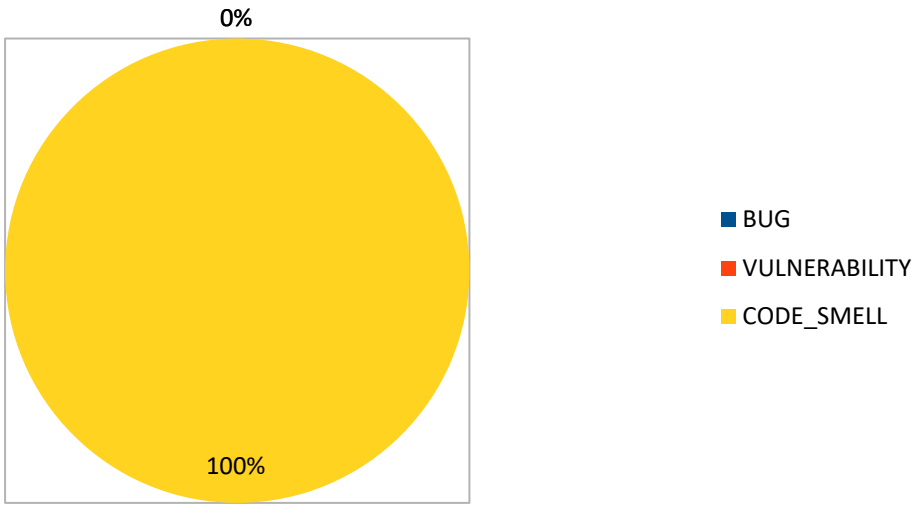
ISSUES

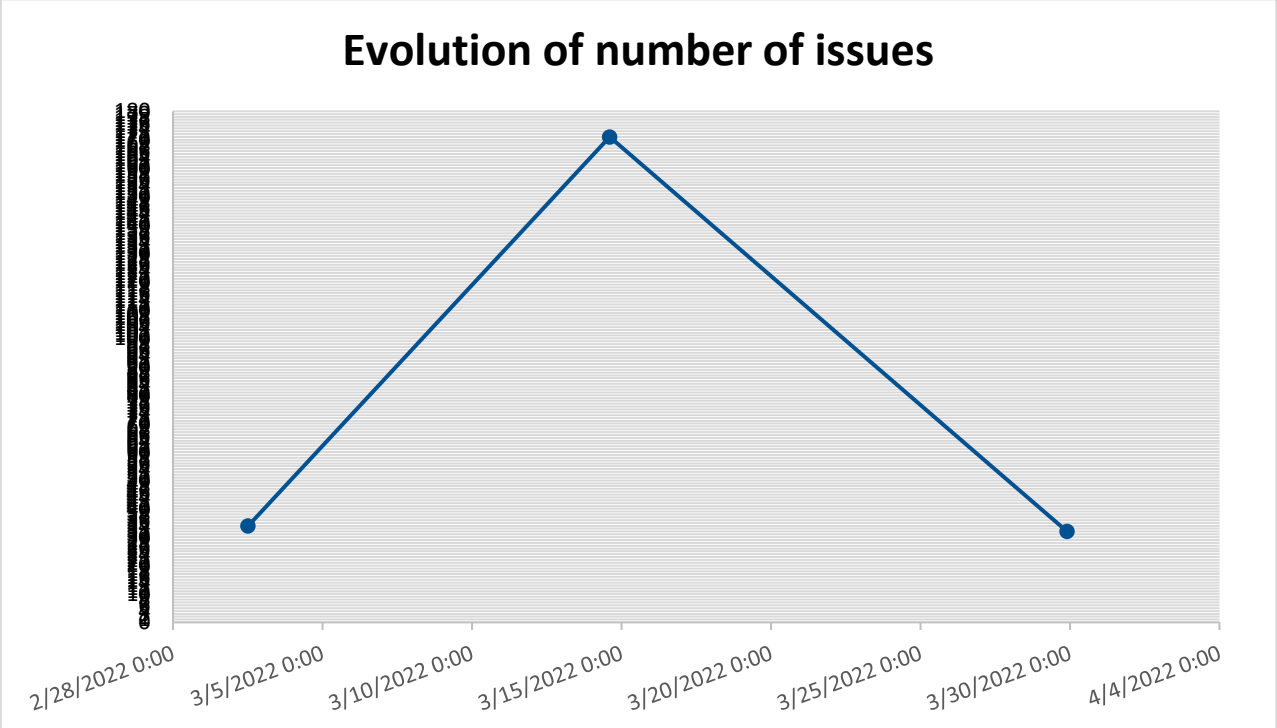
CHARTS

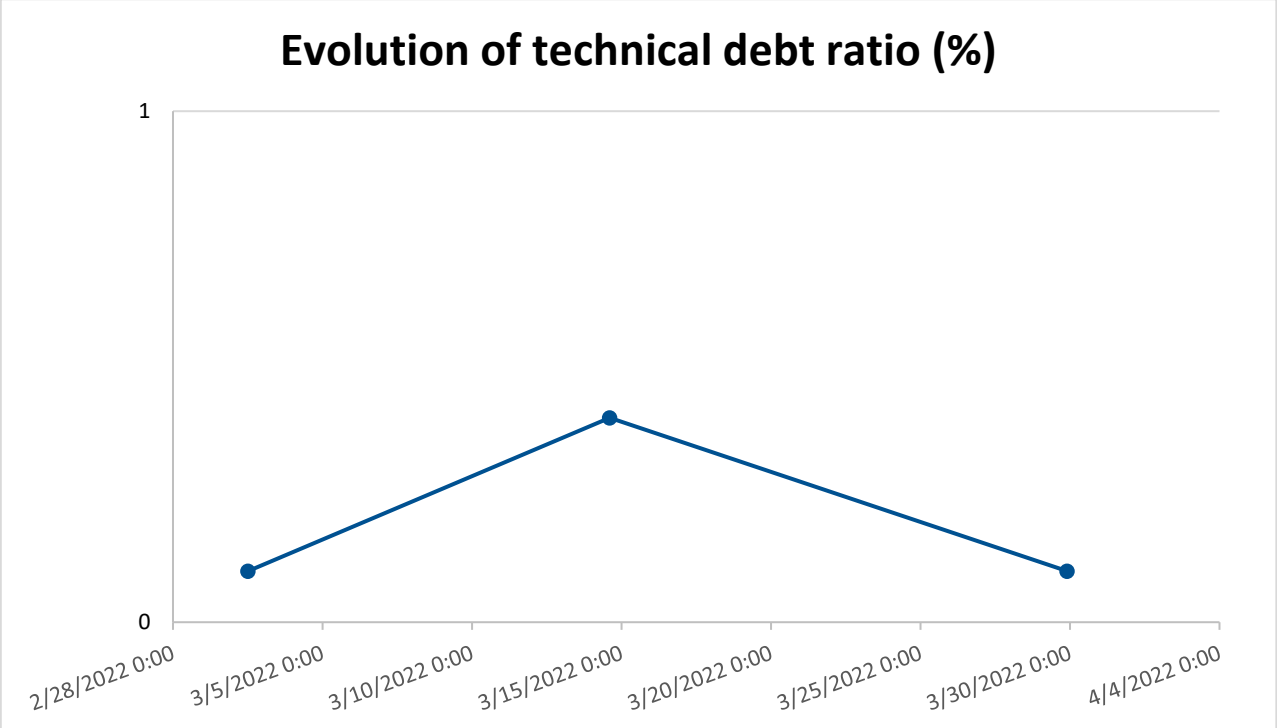
Number of issues by severity



Number of issues by type




Evolution of number of issues
2/28/2022 0:00
3/5/2022 0:00
3/10/2022 0:00
3/15/2022 0:00
3/20/2022 0:00
3/25/2022 0:00
3/30/2022 0:00
4/4/2022 0:00


Evolution of technical debt ratio (%)
1
0
2/28/2022 0:00
3/5/2022 0:00
3/10/2022 0:00
3/15/2022 0:00
3/20/2022 0:00
3/25/2022 0:00
3/30/2022 0:00
4/4/2022 0:00

ISSUES COUNT BY SEVERITY AND TYPE

Type / Severity	INFO	MINOR	MAJOR	CRITICAL	BLOCKER
BUG	0	0	0	0	0
VULNERABILITY	0	0	0	0	0
CODE_SMELL	0	28	4	0	0

ISSUES LIST

Name	Description	Type	Severity	Number
Source files should not have any duplicated blocks	An issue is created on a file as soon as there is at least one block of duplicated code on this file	CODE_SMELL	MAJOR	3
Unused function parameters should be removed	Unused parameters are misleading. Whatever the values passed to such parameters, the behavior will be the same.	CODE_SMELL	MAJOR	1
The default unnamed package should not be used	According to the Java Language Specification: Unnamed packages are provided by the Java platform principally for convenience when developing small or temporary applications or when just beginning development. To enforce this best practice, classes located in default package can no longer be accessed from named ones since Java 1.4. Noncompliant Code Example public class MyClass { /* ... */ } Compliant Solution package org.example; public class MyClass{ /* ... */ }	CODE_SMELL	MINOR	1
Unnecessary imports should be removed	The imports part of a file should be handled by the Integrated Development Environment (IDE), not manually by the developer. Unused and useless imports should not occur if that is the case. Leaving them in reduces the code's readability, since their presence can be confusing. Noncompliant Code Example package my.company import kotlin.String // Noncompliant; "kotlin" classes are always implicitly imported import my.company.SomeClass // Noncompliant; same-package files are always implicitly imported import java.io.File // Noncompliant; "File" is not used import my.company2.SomeType class ExampleClass { val someString = "" val something =	CODE_SMELL	MINOR	1

	SomeType() } Compliant Solution package my.company import java.io.File import my.company2.SomeType class ExampleClass { val someString = "" val something = SomeType() lateinit var fileUsage: File } Exceptions Imports for types mentioned in KDoc comments are ignored.			
Code annotated as deprecated should not be used	Code annotated as deprecated should not be used since it will be removed sooner or later. Noncompliant Code Example @Deprecated("") interface Old class Example : Old // Noncompliant	CODE_SMELL	MINOR	26

SECURITY HOTSPOTS

SECURITY HOTSPOTS COUNT BY CATEGORY AND PRIORITY

Category / Priority	LOW	MEDIUM	HIGH
LDAP Injection	0	0	0
Object Injection	0	0	0
Server-Side Request Forgery (SSRF)	0	0	0
XML External Entity (XXE)	0	0	0
Insecure Configuration	2	0	0
XPath Injection	0	0	0
Authentication	0	0	0
Weak Cryptography	0	0	0
Denial of Service (DoS)	0	0	0
Log Injection	0	0	0
Cross-Site Request Forgery (CSRF)	0	0	0
Open Redirect	0	0	0
SQL Injection	0	0	0
Buffer Overflow	0	0	0
File Manipulation	0	0	0
Code Injection (RCE)	0	0	0
Cross-Site Scripting (XSS)	0	0	0
Command Injection	0	0	0
Path Traversal Injection	0	0	0

HTTP Response Splitting	0	0	0
Others	3	0	0

SECURITY HOTSPOTS LIST

Category	Name	Priority	Severity	Count
Insecure Configuration	Delivering code in production with debug features activated is security-sensitive	LOW	MINOR	2
	Using clear-text protocols is security-sensitive	LOW	CRITICAL	1
Others	Broadcasting intents is security-sensitive	LOW	CRITICAL	1
Others	Receiving intents is security-sensitive	LOW	CRITICAL	1
Others	Allowing application backup is security-sensitive	LOW	MAJOR	1